AR-006-979

DSTO AUSTRALIA

ELECTRONICS RESEARCH LABORATORY

Information Technology Division

REPORT
ERL-0637-RE

RECOMMENDATIONS FOR THE USE AND TAILORING OF DOD-STD-2167A

by

Andrew P. Gabb, Peter C. Pollard
Stefan F. Landherr, Rudi J. Vernik

SUMMARY

This report is the culmination of a study into the use of DOD-STD-2167A in Australian software development projects. It makes recommendations for the use and tailoring of the standard.

SEP 92

APPROVED FOR PUBLIC RELEASE

POSTAL ADDRESS: Director, Electronics Research Laboratory, PO Box 1500, Salisbury, South Australia, 5108.

CONTENTS

ABBREVIATIONS

4GL	Fourth generation language
CALS	Computer-aided Acquisition and Logistics Support
CASE	Computer aided software engineering
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CIDS	Critical Item Development Specification
COTS	Commercial off the shelf (ie non-developmental)
CRISD	Computer Resources Integrated Support Document
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSOM	Computer System Operator's Manual
CSU	Computer Software Unit
DID	Data Item Description
DSTO	Defence Science and Technology Organisation
FSM	Firmware Support Manual
HWCI	Hardware Configuration Item
IDD	Interface Design Document
IRS	Interface Requirements Specification
PDR	Preliminary Design Review
PIDS	Prime Item Development Specification
RFT	Request for Tender
QA	Quality Assurance
SDD	Software Design Document
SDF	Software development files
SDP	Software Development Plan
SOW	Statement of Work
SPM	Software Programmer's Manual
SPS	Software Product Specification
SRS	Software Requirements Specification
SSDD	System/Segment Design Document
SSR	Software specification review
SSS	System/Segment Specification
STD	Software Test Description
STP	Software Test Plan
STR	Software Test Report
SUM	Software User's Manual
UI	User interface
V&V	Verification and validation
VDD	Version Description Document

EXECUTIVE SUMMARY

DOD-STD-2167A is a U.S. military standard establishing requirements for software development. Much of the operational software currently being developed for the Australian Department of Defence is being developed in accordance with this standard; its use is likely to be mandatory in almost all future operational systems.

The transition to this new standard has not been straightforward. There have been numerous criticisms about DOD-STD-2167A particularly with regard to the amount of documentation required and the effort needed to produce it. Doubts have also been raised as to the actual value of such documentation to the customer or developer. These problems are not unique to 2167A, however. Similar problems were experienced with earlier standards, but 2167A is affecting more projects and developers and thus is perceived as a larger problem.

This report is the culmination of a study into the use of 2167A in Australian software development projects. A prior report addressed the problems experienced in the use of the standard in this country.

The study was planned to be as open as possible. In addition to a comprehensive survey of 2167A policy and usage in the early stages, drafts of both reports were widely distributed to all interested parties and workshops were held in Canberra and Adelaide to discuss the findings.

DOD-STD-2167A is used in projects because it is the prescribed standard for most Defence software developments in Australia. In some cases customers outside Defence specify it because it is the only well known comprehensive standard available.

It is not simple to use - developers with less structured development environments find it particularly difficult to understand. It also leans heavily on other military standards which must also be understood if it is to be used effectively. In some cases there are conflicts between different standards which make the task more difficult.

DOD-STD-2167A is neither perfect nor complete. It contains contradictions and omissions which must be rectified by tailoring the standard. Successful use of this standard is critically dependent on the quality of the tailoring. Tailoring must be applied for *all* projects, and whenever possible, tailoring should be carried out as a joint effort between customer and developer.

Most problems with 2167A stem from the following:

- The standard is poorly understood by either the customer or developer or both.
- The standard is often inadequately tailored to meet the needs of the project.
- The developer does not have a systematic development process.
- There is insufficient meaningful communication between the customer and developer.

Not surprisingly, the following principles for using 2167A are recommended:

- Provide appropriate education and training for all staff involved in the use of 2167A.
- DOD-STD-2167A *must* be tailored.

- Special consideration needs to be given to enhancing the communication and cooperation between customer and developer.

- The development methodology selected for the project must match the requirements of 2167A (appropriately tailored) and must be adhered to.

There is no "silver bullet" for tailoring. Tools and guidelines are available but they are no substitute for knowledge and experience. They merely assist in the process of tailoring, but not in making the hard decisions which will arise. Tailoring must also consider the family of standards to be used, not DOD-STD-2167A in isolation. Guidelines in 2167A and its tailoring handbook also restrict the scope of tailoring to less than what is necessary in most circumstances, reducing the advantages that tailoring can achieve. There are numerous occasions when the tailoring should exceed that suggested by the guidelines.

It appears to be popular to criticise 2167A for its dependence on the "waterfall" model of software development - to claim that it is outdated and inappropriate for "modern" development methodologies. The authors suggest that many such criticisms are based on an inadequate understanding of 2167A, often coupled with an unwillingness to impose systematic control on the development process. An inability or unwillingness to apply appropriate tailoring is also a barrier to adapting 2167A to different models.

Specific recommendations are made for reducing the amount of documentation, streamlining reviews and audits and identifying the relationship of the standard to testing and configuration management.

The study has also identified areas for further research in the development and acquisition of Defence software in Australia. These include the assessment of the capability of developers and customers to participate in 2167A projects, problems in the preparation of RFTs and tenders for software intensive projects, the use of electronic documentation, the establishment of 2167A resource repositories and the application of verification and validation (V&V) to Defence projects.

This report offers advice and recommendations which enhance the understanding and use of DOD-STD-2167A, and which are already helping to improve the development of software for Defence projects. The most important recommendation, however, is that both customers and developers must dedicate more effort to the education of their staff - DOD-STD-2167A is a standard which does not forgive ignorance or amateurism.

Copies of this document on magnetic media are available from the authors on request.

1 INTRODUCTION

1.1 Purpose

DOD-STD-2167A is a U.S. military standard establishing requirements for software development. Much of the operational software currently being developed for the Australian Department of Defence is being developed in accordance with this standard; its use is likely to be mandatory in almost all future operational systems.

The transition to this new standard has not been straightforward. There have been numerous criticisms about DOD-STD-2167A (and its precursor DOD-STD-2167), particularly with regard to the amount of documentation required and the effort needed to produce it. Doubts have also been raised as to the actual value of such documentation to the customer or developer.

This report is the culmination of a study into the use of DOD-STD-2167A in Australian software development projects. It makes recommendations for the use and tailoring of the standard.

As the initial phase of this study the authors conducted a survey of 2167A policy and usage in software development projects [GAB91]. Although the study was primarily aimed at defence projects, there are several non-defence applications of the standard in Australia, both in internal developments and in commercial applications. These were also considered in the survey.

A draft of this report was circulated to interested parties for comment and was the subject of workshops held in Canberra and Adelaide in April/May 1992. This final report includes feedback from these activities.

The opinions expressed in this paper are those of the authors and do not represent the policy or official standpoint of DSTO or the Department of Defence.

1.2 Scope

This report provides discussion about and recommendations for the use and tailoring in DOD-STD-2167A projects. Much of the material is based on the assumption that the reader is generally conversant with the requirements of 2167A, and that the software development occurs as the result of a contract between an organisationally distinct customer and developer.

Initially it was assumed that the major cause of difficulties in the use of 2167A in Australia was in the complexity of the tailoring activity and it was therefore intended that this study would concentrate mainly on the tailoring of 2167A. The survey indicated that tailoring was only part of the problem and the scope of the study was broadened to encompass the more general issues of 2167A use. Consequently, less effort has been directed towards providing detailed guidelines for tailoring.

Similarly, this report does not address other aspects of software development projects which some readers may regard as critical. One example is the relationship of 2167A to quality standards such as DOD-STD-2168, AS 3563 and AS 3901.

1.3 Nomenclature

The terms "customer" and "developer" are used in this paper to indicate those responsible for software system procurement and development respectively. In some cases the terms are used

to indicate individuals in the customer and development teams rather than the organisation that they represent. The terms "users" and "maintainers" are used to indicate those who will use the software (the operators for example) and those who will maintain it in the customer's organisation.

1.4 Organisation of this report

Section 2 examines the purpose for 2167A, some of its deficiencies and experiences with its use in Australia.

Section 3 recommends a basis for the use of the standard - advice to customers and developers in preparing to use the standard.

Section 4 discusses the general principles of tailoring 2167A and related standards.

Sections 5 and 6 address the general requirements of 2167A and the allocation and partitioning of the basic software elements, Computer Software Configuration Items (CSCIs), components (CSCs) and units (CSUs).

Section 7 addresses the development process and development methods, their relationship with 2167A, and possible changes to the process, methods or the standard that may be necessary.

Sections 8 and 9 examine the documentation issues, firstly in a general sense, then for individual documents.

Finally, Sections 10 and 11 discuss the necessity of further work and the conclusions of the study.

1.5 Acknowledgments

The authors thank all those in industry, academic institutions and the Department of Defence who have given their time and assistance in contributing to this study. We wish to acknowledge the assistance of the following organisations:

Ansett Technologies
Australian Defence Industries Ltd
Australian Submarine Corporation Pty Ltd
AWA Defence Industries Pty Ltd
BHP Information Technology Ltd
British Aerospace Australia Ltd
CAA Systems Support Group
Compucat Research Pty Ltd
Computer Sciences of Australia Pty Ltd
Ferranti Computer Systems (Australia) Pty Ltd
Hawker De Havilland Victoria Ltd
Kinhill Engineers Pty Ltd
Lasotell Pty Ltd
Logica Pty Ltd
RADE Systems Pty Ltd
Rockwell Ship Systems Australia
State Rail Authority of NSW
Technology Australia Pty Ltd
Telstar Systems Pty Ltd

Tripal Systems Pty Ltd
TRW Systems Integration Group
Wormald Advanced Systems Engineering

2 DOD-STD-2167A - THEORY AND PRACTICE

2.1 The reasons for using DOD-STD-2167A

DOD STD-2167A is usually used in projects not by choice but because it is the prescribed standard for software development. The developer uses it because it is specified in the contract; the customer specifies it either because it is the customer's prescribed policy to do so, or because there are no other reasonable alternatives. There are of course many standards which address the issues of software development, but none which covers the same breadth with the same levels of prescription as this standard.

One advantage of using 2167A is that it is part of a (very large) standards family which should guarantee its consistency with other standards. It meets this objective reasonably well, although not without some problems (See section 4.4). Requirements not defined in 2167A are generally provided in other related military standards, reducing the likelihood of omissions and conflicts.

It also provides a consistent basis for management of a software development project by the customer. As MIL HDBK 287 describes,

> *It establishes standard terminology, provides a standard set of deliverables, reviews and audits to choose from, and defines a standard set of software management practices that may be imposed.*

Visibility of the development is assured not only by the reviews and audits, but also by the detailed requirements for the format and the content of the delivered documentation. The system of reviews also provides increasing visibility as the design becomes more detailed, allowing the customer to assess both the effect of design choices on the operational requirement and the developer's ability to complete the development successfully.

Finally, it imposes standards for the development process that are arguably higher than those that many contractors might employ if not obliged otherwise. In this way the customer tries to guarantee a reasonable level of quality for the product.

These reasons are important in understanding how 2167A should be used and how, or if, it should be changed (tailored) for a particular project.

2.2 *Relationship to other standards*

DOD-STD-2167A has been designed to be part of a large family of integrated military standards, and it assumes that several of these standards will also be specified when using 2167A, providing an integrated framework for software development. If one or more of these standards are not specified, 2167A must be tailored both to remove the references to them and to compensate for the fact that their requirements are no longer included. The relevant standards are:

THIS
PAGE
IS
MISSING
IN
ORIGINAL
DOCUMENT

effect of the tailoring on the remainder of the standard, resulting in conflicts with other parts of 2167A and with other standards.

Perhaps more serious is the authors' experience that, whether 2167A is tailored or not, contractors rarely meet their full obligations with regard to the standard. In these cases the shortfalls are often overlooked by the customer's monitoring team or not enforced when detected.

There is also evidence that both customers and developers concentrate their attention on the documentation deliverables at the expense of the development requirements.

Developers' criticisms of the "inappropriateness" of 2167A documentation requirements often stem from the lack of a systematic approach to development. DOD-STD-2167A requires a systematic development method and its documentation requirements reflect this. Problems arise when developers attempt to document a poorly structured existing design using the 2167A formats [MCG90]. More generally, 2167A specifies activities and provides a strict framework for their documentation. If the activities are not performed, or are not carried out in the required manner or order, the documentation process is difficult and the resulting documentation is likely to be of low value.

It is interesting to note that while developers are concerned about the amount of documentation required, many customers believe that developers tend to provide too much data in requirements and design documents, while omitting critical information. It appears that some developers anticipate that a customer is less likely to reject a thick document, perhaps based on previous experiences. The authors have observed some evidence of truth in this belief.

Many projects have seemingly been burdened by a lack of understanding and experience with 2167A on the part of the developer, the customer, or both. Developer inexperience has resulted in difficulties in developing software and its associated documentation to the required standard. Customer inexperience has resulted in inadequate and inconsistent tailoring, and an inability to adequately monitor the development and review delivered documentation. The inevitable result of inexperience is that either or both of the parties are unable to identify limitations in the development process or areas of potential risk.

One of the side effects of developer and customer inexperience is that some developers do not assess the full costs of using 2167A in tenders for software projects, either through lack of understanding of the standard, or on the assumption that the customer will not enforce what are in fact contractually binding requirements. Not only is this unfair to developers who include the full cost of 2167A development in their tenders, but it also adds risk to the project - either in the quality of the product if the full requirements are not met, or in the developer being obliged to meet requirements for which he has not budgeted.

2.6 Omissions in 2167A

This section addresses perceived omissions in 2167A and is mainly concerned with the documentation of commonly needed information. These omissions should normally be rectified by either tailoring existing requirements or by the specification of additional requirements in the SOW (Statement of Work) or CDRL (Contract Data Requirements List).

2.6.1 System design

A notable omission from 2167A is the lack of a requirement for an overview of the software system design, mapping the requirements to the design and illustrating how the functions of the

system are met in terms of execution and data flow (at a relatively high level) [MAR88, SPR89]. Such information is extremely useful in understanding how a complex system works and for tracing the source of defects during maintenance. To some extent this information is included in the Software Design Document (SDD) in paragraph 3.1 with regard to "states and modes" but it is inadequate in its level of detail and description.

For a system of several interacting CSCIs there is no obvious place for such a description, apart from the System/Segment Design Document (SSDD) which is at too high a level and is delivered too early in the project. In this case one or more additional documents are needed, possibly called "System Architecture and Design Description", coordinating the design of the CSCIs. The authors see such a document being developed incrementally in a manner similar to SDDs, with preliminary information being provided at the Preliminary Design Review (PDR).

2.6.2 Documentation overview and glossary

In larger projects, an overview of the interaction between the different specifications and other documents is required, to allow specialists and casual project staff to find specific information. A project-wide glossary of abbreviations and definitions is also often essential, both to assist newcomers and to maintain a consistent nomenclature throughout the project.

2.6.3 User interface design

One difficulty that several developers have faced with 2167A is its lack of guidance with respect to the documentation of the design of the system's user interfaces (UIs) [MEY89]. The most obvious place for this information is in the Interface Design Document (IDD) which does not, however, provide appropriate formats for its definition. The Software User's Manual (SUM) is intended eventually as a manual for the system and will be derived from the UI design. However it is developed far too late in the process, and in most projects would not contain all the design information needed.

UI design should occur early in the development process and be reviewed at PDR [1521B]. Where there is a complex UI its design should be specified in a separate document (the "User Interface Design Document" is an apt name); if the UI is simple it may be included in a tailored IDD. MIL-STD-7935A's requirements for its "Users Manual" and "End User Manual" may offer assistance in determining the contents of such a specification [7935A].

2.6.4 Customer interaction in requirements refinement and design choices

In many projects the refinement of requirements and detailed design can lead to implementations that the customer regards as unsatisfactory. This is particularly likely in areas such as the definition of the user interface and in the specification of detailed performance (such as response times). More importantly, although the implementation may be unacceptable, it is often either compliant with the higher level requirements or the judgement of compliance is a subjective issue. While it might be claimed that such a situation is a result of poorly specified requirements, this will frequently not be the case. Customers are encouraged to avoid detail in requirements that might inhibit the design (see also Section 9.2). The penalty for lack of detail is a development resulting in an unacceptable design.

It is obvious that, regardless of whether the design is compliant or not, the cost of rectification will be minimised if the deficiency is detected early. The review process required by 2167A and 1521B is insufficiently responsive to solve this problem, and is likely to result in either additional costs borne by the customer or developer or both, or the deficiencies not being rectified.

In most projects the likelihood of such a situation arising can be predicted and catered for in the development process, particularly if it is identified as a risk area (which it is). DOD-STD-2167A addresses this problem only broadly, requiring that the developer identify areas of "potential technical, cost or schedule risks" [2167A 4.1.4]. The SOW should include the requirement for the developer to additionally identify areas of potential disagreement or subjective interpretation and to propose procedures to resolve these as early in the development process as possible. These might include for example user interface prototyping or the use of an incremental development process.

2.6.5 Software representation and generation

The design of each CSCI will often result in specific decisions being made with regard to the representation of the software (such as source file partitioning and naming) and the environment in which the software is to be generated, including the use of standard libraries, compiler directives and options for the use of the compiler and linker. Where these decisions are made as part of the detailed design process, and are therefore relevant to the coding, integration and testing activities, they should be documented as part of the design.

This information should be documented in the SDD for each CSCI, CSC or CSU as applicable.

2.6.6 Traceability requirements

DOD-STD-2167A specifies that requirements are to be traceable from the high level specification (eg SSS) to CSCIs, CSCs and CSUs, and from the CSUs to the SRS and IRS. The DIDs, however, require this information in a different form, providing traceability:

- From the SSDD to the SSS (in the SSDD)
- From the SRS to the SSS, CIDS or PIDS (in the SRS)
- From the SSS, CIDS or PIDS to the SRS (in the SRS)
- From CSUs to the SRS and IRS (in the SDD).

This provides traceability from CSUs to the high level specification in an indirect form (through the SRS). To ensure that the high level requirements are met, it is more useful to have a traceability matrix from the high level specification directly to the CSCIs, CSCs and CSUs (and to the IDD if necessary). For larger projects this would require a separate document used solely for showing traceability. Where automated traceability tools are used this should entail little extra effort.

3 BASIS FOR USING 2167A

This section provides observations and recommendations for using DOD-STD-2167A.

The prime recommendation (which might appear to be obvious) is that both customer and developer must have a detailed understanding of, and preferably experience in, the requirements and purpose of DOD-STD-2167A and, when specified, the related military standards.

3.1 Education and training

To achieve a thorough understanding of the purpose and requirements of DOD-STD-2167A and related military specifications is not an easy task and requires considerable education and training. Tertiary computing qualifications are desirable but by themselves are not enough, since graduates are not adequately prepared in these areas. Experience in real software

engineering is critical, especially exposure to working with standards. For a successful project both customers and developers must have the appropriate levels of knowledge and experience [MCG90, MEY89, SPR90].

Ideally, all development staff should have several year's experience in software engineering (preferably in the same application domain as the project), should have experience in the programming language, development method and development environment to be used, should have been trained in using 2167A and should have used the standard in a previous development task. This happy state is rarely achievable. The technical software manager and the senior designers should have this level of expertise, however, before the project starts. Other members of the development team must have appropriate computing qualifications, and must be given appropriate training in the language, methods and environment, and in the use of relevant standards including 2167A.

The customer's technical team for a large software project should have at least one person with the level indicated above for the developer's senior designers. It may be adequate in small to medium software projects for the project office to rely on external expert advisers, but at least one person in the day to day management of the project must have experience and understanding of the use of the relevant standards and software engineering management and methods.

In some cases, the developer may need to educate the customer, particularly in the development method and CASE tools used. This may involve additional cost to the project, in both time and money.

The authors recognise that the pool of experienced software engineers in Australia is limited by the relatively small size of the defence/aerospace software market and that low mobility exacerbates the problem. Nevertheless, developers and customers must place a *high priority* on the recruitment and retention of appropriately skilled staff.

3.2 Communication and cooperation

There are many reasons supporting a close *working relationship* between developer and customer [SPR90], but the authors are aware of few projects where such a relationship exists. In some cases the customer's team does not have the experience to support such a relationship and the developer is understandably reluctant to provide the needed education [MEY89]. Developers are also wary of the visibility that informal communication might bring, and the potential for subsequent criticism of their development process.

The benefits of increased communication and cooperation between developer and customer are seen to be:

a. Better mutual understanding of each others' problems and a subsequent increase in mutual trust. The benefits of this will also be shown by more meaningful and more successful negotiations and reviews [FIS87].

b. Improved risk management. Risk areas can be highlighted earlier and may be assessed by both customer and developer before a solution is proposed.

c. Fewer misunderstandings and misinterpretations. Problems are identified earlier and, when they cannot be resolved informally, may be flagged for prompt formal resolution.

d. Decreased development effort because of the reduced likelihood of a course of action being rejected by the customer.

e. Higher quality documentation (at least from the customer's point of view). Discussions should establish the form of documentation and the level of detail that the customer requires.

The risk of poor communication will be higher when there are many contractors developing software, and where the prime contractor is not a major software developer. In these cases special consideration needs to be given to enhancing the communication between the customer and the actual developer of the software.

Meyer et al. [MEY89] state:

> *The consistent trust and goodwill of the customer towards your design effort will be the single most important factor in your ability to complete your project on time and within budget.*

3.3 Development methodology

In software engineering, the term "methodology" refers to a combination of life-cycle models (paradigms), management practices, technical practices, tools and training procedures used to produce software [DEG90]. There are many different models, practices, tools and procedures to choose from, so there are many different development methodologies, each with certain strengths and weaknesses.

The methodology selected for a particular software development project must be suited to the application domain, the implementation language, the magnitude of the development effort and other characteristics of the project and the development organisation. In addition, if the development is to be conducted in accordance with DOD-STD-2167A then the development methodology must map sensibly into 2167A. With appropriate tailoring, 2167A can be forced to accommodate almost any methodology, but it is pointless to force fit a methodology that does not support the basic principles of 2167A (orderly development process, visibility, reviews and audits) or that is not suitable for programming in the large.

The development methodology must be selected quite early in the project (such as at tender preparation time, or equivalent for in-house projects). The methodology must be written down and it must be approved by the customer before development begins [MEY89]. The appropriate place for specifying the methodology (usually by referencing separate documents) is in the preliminary Software Development Plan (SDP). Where a significant amount of time elapses between tender and contract award, it may be necessary to redefine the development methodology during the contract negotiation activities.

Clearly the developer's staff must be well-versed in the methodology and there should be a training plan to ensure that they are. It is also very important that the customer's technical staff have an understanding of the methodology, and this may well require training for them also.

This issue is discussed in further detail in Section 7.

4 PRINCIPLES OF TAILORING

4.1 General

The main reasons for tailoring are as follows:

- To reduce the development effort required and hence to reduce the cost and time to completion.
- To modify or add requirements which are needed for a particular project or software development method.
- To improve the quality of the product and the documentation.
- To correct inconsistencies and hence to reduce risk in the contractual process.

In the authors' opinion, tailoring is essential for all 2167A projects.

Tailoring requires a good understanding of the purpose and details of DOD-STD-2167A and experience in software engineering practices and management. Tailoring for a specific project requires further understanding of the application, the project requirements and (for detailed tailoring) the development method used. A comprehensive understanding of other standards (particularly MIL-STD-1521B) is also required if these standards are to be used in the project.

Tailoring is difficult, time consuming and requires a meticulous approach. It is essentially an iterative process - each proposed change must be assessed with regard to other changes and its effect on the management and development process as defined in DOD-STD-2167A and other relevant standards.

Uncontrolled, simplistic or inexperienced tailoring is likely to lead to potentially serious problems as follows:

- Loss of needed visibility to the customer.
- Omission of critical requirements.
- Inconsistencies between requirements, including those in other standards.
- Documentation which is inadequate or of poor quality.

Tailoring should be refined throughout the course of a project [248A, 287]. The Request for Tender (RFT) should address the high level tailoring of all relevant standards and specifications based on the standards chosen, the application and the acquisition process. The RFT should also state the customer's tailoring policy and solicit recommendations from tenderers with regard to cost effective tailoring. The detailed tailoring should be agreed during contract negotiations and included in the contract. Further tailoring will often be necessary during the course of the contract. In cases where the need for refinement during the development can be predicted, plans for modifying the tailoring should be included in the SDP.

It is recommended that, where possible, tailoring is carried out as a joint effort between the customer and developer [287]. This will ensure that each is aware of the justification and consequences of each individual tailoring, as well as encouraging an atmosphere of communication and cooperation (see also Section 3.2).

4.2 Tailoring references and tools

General principles for the tailoring of military standards and specifications are addressed in DOD-HDBK-248A, which discusses the advantages, dangers and mechanisms of tailoring. MIL-HDBK-287 provides a more detailed approach to tailoring DOD-STD-2167A, and provides good advice on the relationship of 2167A to other standards and the handling of the "shell" requirements. Both of these handbooks are recommended to those responsible for tailoring and in particular to customers responsible for the approval of tailoring.

Logicon's TAILOR computer based tools [LOG90] are also useful, especially for the documentation of the tailoring and the preparation of CDRLs. They have been found to be particularly useful in collaborative tailoring, where the customer and developer draft the tailoring together, using the tools as a basis. These tools are mainly aimed at tailoring for the full scale development phase of a project and are less useful for the tailoring of investigatory or research projects.

4.3 Project factors influencing tailoring

Each project is different. DOD-STD-2167A provides requirements which are applicable for many projects, but the requirements should be tailored to meet specific project factors. Factors which may influence tailoring include:

- The general application area for the software, eg operational, prototype, feasibility demonstration, analysis, test or training.
- The type of software being developed - management information systems and real-time combat systems, for example, should require different tailorings.
- The development method and CASE tools used.
- The use of non-procedural languages or application specific program generation tools (such as 4GLs).
- The difficulty or complexity of the task.
- The number and complexity of external and inter-CSCI interfaces.
- Whether parts of the software are safety critical or security critical.
- The proportion of software which is reused from a previous project and the amount of non-developmental and COTS (commercial off the shelf) software.
- The level of software support that will be required either by the customer or developer.
- The verification and validation (V&V) approach, including whether this function will be performed internally or by an independent or customer based V&V team.

4.4 Staying consistent with other standards

DOD-STD-2167A, as it stands, is inconsistent with some U.S. military standards ([287 Appendix B]). Tailoring 2167A and/or the standards in conflict is required to resolve these inconsistencies. In particular MIL-STD-1521B was written to correspond to DOD-STD-2167

and has numerous references which are incorrect when this standard is used with DOD-STD-2167A.

Each change to 2167A or the Data Item Descriptions (DIDs) must be considered not only with respect to the internal consistency of 2167A, but also with respect to other required standards.

The problem of inconsistency between standards will always be a contractual risk area particularly when standards are released at different times. Where possible, customers should specifically nominate the version of standards which are to apply to reduce misunderstandings in this area.

An understanding of the value and purpose of each standard specified is important in preventing inconsistencies. Customers should avoid the "shotgun" approach of specifying as many standards as possible in the hope that this will provide added protection. Not only is this likely to guarantee inconsistencies between the standards, but may also lead to additional costs.

MIL-STD-1521B is likely to present the most problems regarding conflicts. The sequence of activities, documentation delivery and reviews of 2167A are tightly interwoven with the requirements of 1521B. Deferring the delivery of a document to a later stage of the development, for example, may mean that it is not available for review as required by 1521B, and that there are no requirements to review it at later reviews. Deleting the requirements for a particular review may result in some activities or documents not being reviewed at all.

4.5 Add, delete or modify?

DOD-STD-2167A (para. 1.3) states that "the tailoring process intended for this standard is the deletion of non-applicable requirements". MIL-HDBK-287 (para. 4.3.1.a) states that "For DIDs, requirements may be deleted or partially deleted, but not modified". Neither of these restrictions is binding on Australian users of 2167A and it is evident that in many cases an adequate tailoring will not be possible without the modification of requirements, including those for the DIDs [MAR88, MCG90].

The authors therefore recommend that the tailoring of 2167A and its DIDs should be accomplished by the deletion, addition and modification of requirements. For consistency between Defence projects however, it is important that the structure and paragraph numbering of both the tailored requirements and the required documentation be preserved as far as possible.

4.6 Over-detailed tailoring

It is not necessary to tailor out paragraphs which are obviously inapplicable to individual software elements (eg particular CSUs) but which may be applicable to others. Multiple or over-detailed tailorings are difficult for documenters to follow, difficult for those refining the tailoring to maintain, and are prone to error. It is preferable that the tailoring concentrates on the major variations from the requirements of 2167A rather than the recording of petty changes which will have no effect on the quality of the software and its documentation, or the effort required in producing these.

4.7 The cost of tailoring

The authors believe that there is no cheap solution to the tailoring problem. Projects have widely differing requirements with regard to both the development process used and the

documentation deliverables. These differences must be accommodated by tailoring to reduce the cost, schedule and risk and to improve the quality of the product, including documentation.

The tailoring of 2167A and its DIDs requires high quality staff with specialised knowledge and experience, staff whose services are normally in strong demand. The task of tailoring is normally time-consuming and may require weeks rather than days over the life of the project.

In the authors' opinion, however, the benefits of an optimal tailoring are likely to far outweigh the costs for all but the smallest projects.

5 THE REQUIREMENTS OF 2167A

This section discusses the use and tailoring of the requirements of 2167A, excluding those for documentation (DIDs).

5.1 Is it all needed?

The full requirements of 2167A can be very costly to implement, not only in the development effort but also through the monitoring task of the customer. Many activities such as product evaluations require additional effort both in their execution and in their recording and auditing.

In addition to the general guidelines for tailoring [287, 248A], the authors suggest that where the developer considers that a requirement is not cost effective and where the customer (or his representative) does not have the means or ability to verify that the requirement is met, consideration should be given to the deletion of that requirement. In the authors' experience, developers are (understandably) reluctant to meet requirements that they consider not to be cost effective and the subsequent implementation is often inadequate. If the customer is not prepared or able to monitor and enforce the fulfilment of the requirement, there is no advantage to the customer or developer in specifying it, and there is little point in the customer paying the additional costs involved.

5.2 Reviews and audits

Reviews and audits as specified in MIL-STD-1521B are the milestones and often the crisis points of 2167A projects. The authors' survey and workshops [GAB91] did not cover the issue of reviews and audits in any detail (an oversight), but revealed concern regarding the overlap and inconsistencies between 2167A and 1521B. Many of the problems that both customers and developers had experienced with 2167A culminated in difficulties with customer approval at reviews, typically Preliminary and Critical Design Reviews (PDRs and CDRs).

It is evident that in many projects reviews are the main occasions when technical discussion about software takes place between the customer and developer. This factor is likely to be responsible, at least in part, for what appears to be a relatively low rate of outright approval on first presentation in design reviews. The developer prepares for such a review with little knowledge of what the customer is likely to approve. The customer, on the other hand, is often presented with information for the first time (formally or informally). The outcome is almost inevitable:

- The customer's technical representatives are overwhelmed by the new information and time is wasted during the actual review meeting in explanation and education. The fact that the developer has in many cases provided more information (often irrelevant to the review) than is needed aggravates rather than assists in this situation.

- Because the design is new and is being seen for the first time, faults in the design are found.

- Different perceptions of project requirements and the use of 2167A result in protracted discussions with regard to interpretation of requirements.

- Important areas are completely ignored due to lack of time.

- The review results in disapproval or reluctant contingent approval (in the words of 1521B).

Such reviews cannot be regarded as productive. Apart from the fact that much of the discussion is at a lower level than is warranted by the number and status of attendees, the developer has wasted effort in preparing for the review and the schedule is likely to be affected as a result. Moreover, many of the real issues have not been tackled, possibly reducing the overall quality of the product. A lasting side effect is likely to be an increased distrust between the two parties.

The following recommendations are made with regard to reviews and audits.

a. The number of surprises at these events should be minimised. This can only be achieved through greater informal communication between the customer's and developer's technical staff (Section 3.2).

b. Customers should ensure that their technical team is adequate in number, education and experience to liaise effectively with the developer's team and review delivered documentation

c. The reviews should concentrate on higher level issues - interpretation of requirements should have been resolved prior to the review.

d. For large projects, the use of incremental reviews should be considered, reviewing components of the product as they are produced, and summarising the results of these at a (later) formal review meeting.

e. The reviews and audits should be planned and their requirements tailored in accordance with the particular characteristics of the project, including the application, the project size and complexity, and the development method used. The planning should include the number, timing and nature of reviews.

f. Reviews should avoid protracted presentations ("dog and pony shows") of information that is fully documented and concentrate on issues such as the inherent design of the software, work done, progress made and areas of potential risk.

g. The demonstration of user interface mockups and other prototypes, and the capabilities of incremental builds should feature prominently in the appropriate reviews.

In many smaller projects in Australia the complete set of reviews cannot be justified and is not required by the contract. Tailoring 1521B and 2167A to remove the requirements for these reviews is relatively straightforward but gives rise to a common tailoring problem: deleting a review may result in the removal of review requirements that are needed. For example, removal of the Software Specification Review (SSR) removes the requirement to review

qualification and quality factor requirements of CSCIs, which are not subsequently reviewed in the PDR. When reviews are tailored out their requirements should be analysed and those that are required should be added to a subsequent review.

5.3 Testing

DOD-STD-2167A requires testing above the CSU level to be conducted by a team independent of the development team. This requirement entails additional cost and is often not met, particularly in small software development organisations. In the authors' opinion this requirement is important to the quality of testing and its removal can only be justified in exceptional circumstances. It is therefore important that customers (and their V&V agents) pay particular attention to ensuring that the independence of testing is planned and implemented.

The authors have noticed a tendency for some developers to attempt to postpone the definition of formal qualification testing to a later stage than that required by 2167A (in some cases well after CDR). Presumably this is motivated both by a desire to decrease the number of tasks during preliminary and detailed design, and to defer test definition until the performance of the system is known. Whatever the reason, it is neither in the customer's or developer's interests to defer test definition.

Springman [SPR90] indicates one developer's view towards test definition with recommendations including:

- Defining the standards and procedures for testing very early in the contract.
- Involving the customer in developing the test program.
- Getting the customer to commit to a cost effective test program at an early stage.

The benefits to the developer are obvious - the definition (and limiting) of testing at an early stage means that the objectives of the development become clearer and more bounded. From the customer's point of view the test details indicate that the developer understands the requirement and they also should provide a clear indication of the projected performance.

If the developer cannot define adequate formal qualification tests prior to the CDR, it may be an indication of an unstable or insufficiently detailed design.

The authors strongly recommend the definition of formal testing at an early stage in the contract and no later than that specified in 2167A.

5.4 Software product evaluations

There is an evident lack of understanding of the requirements for software product evaluations in current 2167A projects [GAB91], particularly those where these requirements have not been tailored out. This indicates that in many cases product evaluations are not being performed or audited. Product evaluations are conducted by the developer on deliverable software and documentation, prior to their delivery to the customer, to increase the quality of the deliverables. The evaluations must be performed by a team independent of the development team and records must be maintained of the evaluation and the subsequent corrective actions.

There is a temptation for customers to require product evaluations as a form of compensation for the lack of the appropriate skills in their project teams. The authors recommend against this for two reasons. Firstly, the current approach to product evaluations by some developers would

result in no tangible increase in quality. Secondly, a customer not possessing the appropriate skills would not be able to audit the product evaluation process adequately

Product evaluations, if they meet the requirements of 2167A, require a large amount of effort on the developer's part, and an additional auditing effort from the customer. It is likely that experienced developers will perform some form of evaluation in any case as part of their V&V or QA process - it is more cost effective and efficient to detect faults in software and documentation prior to delivery rather than have the deliverables rejected by the customer.

It is therefore recommended that product evaluations be required only on projects (and products) where there are obvious benefits, and where the customer is willing and able to audit the product evaluation process. These should include projects with safety or security critical software, and larger projects where the customer's technical staff cannot monitor all of the development in detail.

5.5 Configuration management

DOD-STD-2167A provides its own requirements for configuration management (although several DIDs refer indirectly to MIL-STD-483). Customers should ensure that the configuration management requirements, including those for post-development maintenance, are compatible with those of 2167A (tailored as needed) and that any specific requirements are also addressed.

Many developers are concerned about the adequacy of their configuration management systems and procedures, particularly with regard to their ability to control more than one version of the software satisfactorily at the same time. Customers should be aware of this problem, particularly for large or complex systems, and be prepared to ensure that the developer's configuration management procedures are both adequate and adhered to.

6 PARTITIONING THE SYSTEM INTO SOFTWARE ELEMENTS

Mistakes made in the partitioning of systems into CSCIs, and the subsequent partitioning of CSCIs into CSCs and CSUs, have been the cause of numerous difficulties in 2167A projects [BUL88, MAR88, MCG90, MEY89]. The partitioning problem is not simply one of providing the most elegant or manageable modularisation based on analysis and design factors. Developers must also consider the effect of partitioning on how they will meet the requirements of 2167A, particularly with regard to review, test and documentation. Defining too many CSCIs will result in too many reviews, formal tests and documents; if the software is insufficiently partitioned it is likely to be difficult to manage and provide insufficient visibility for the customer.

There can be no specific rules for the selection of CSCIs, CSCs and CSUs - each project must make its own judgements in this matter. The following suggestions may be helpful.

6.1 Selecting CSCIs

A CSCI is first and foremost a configuration item. MIL-STD-483A provides sound guidance for the selection of CSCIs and states:

> *The selection of hardware/software to be managed as configuration items should be determined by the need to control a configuration item's inherent characteristics or to control that configuration item's interface with other configuration items. The selection is a management decision normally accomplished through the system engineering*

process in conjunction with configuration management and with the participation of logistics. Selecting configuration items should be with a full view of the life cycle cost and management impacts associated with such a designation. Choosing too many configuration items increases the cost of control; choosing too few or the wrong elements as configuration items runs the risk of too little control through lack of management visibility.

In other words CSCIs need to be selected on the basis of the capability to manage and control them. Consideration needs to be given to the following features:

- Size and complexity - whether the CSCI is too large or the development too complex to be effectively managed.
- Interfaces - CSCIs should be chosen to minimise the interfaces to other CSCIs and HWCIs (Hardware Configuration Items). This is particularly stressed by Buley et al. [BUL88].
- Functionality - functional boundaries, including operational, training and support.
- Criticality - special management may be required for selected elements on the basis of security, safety and other critical factors
- Contractual or geographic issues - it may be unwise for a single CSCI to contain software developed by more than one contractor, at separate locations, or at different stages of a project.
- Post development use and maintenance - whether software is to be used or maintained by separate agencies after development.
- Software environment - whether different elements of the software will run on different processors, for example.
- The likelihood of reuse - if software is likely to be reused in subsequent projects it may be useful to define it in separate CSCIs or CSC's.

Some partitions suggest themselves. A program running on a single processor with reasonably common functions is a natural candidate for a CSCI. If it is very large or there are major divisions in its functionality, consideration needs to given to splitting it into two or more CSCIs. This does not mean, however, that a CSCI should only contain one program Collection of a group of smaller programs with a common function into a CSCI, test and diagnostic programs for example, is sensible practice. It also does not preclude software in different processors, or different processor types, or software written in different languages, or a mixture of new and existing software from being aggregated in a single CSCI under the appropriate circumstances. In this latter case the tailoring for different CSC's in the SDD will need to be considered (see Section 8.3.2).

Both developers and customers should be aiming to minimise the number of CSCIs within the constraints of manageability and visibility.

6.2 Selecting CSC's and CSUs

Selecting the CSC's and CSUs is also a critical activity. If the level at which CSUs are defined is too low, the size and content of the CSCI's Software Design Document (SDD) can be too

large by an order of magnitude. If the level is too high, the customer will be denied the visibility needed to review and approve the design.

A CSC should be regarded as a logical rather than a physical entity - its physical realisation is the set of CSUs which provide its capability (and which may be grouped in sub-level CSCs). While the selection of CSCs will often be a natural outcome of the design method used, consideration of the documentation, integration and testing requirements of the CSCs is important. DOD-STD-2167A requires that each CSC be described in terms of its execution control and data flow (among other things) and that integration and testing within a CSCI be performed on the basis of CSCs. Incorrect selection of CSCs may make these activities quite difficult and time-consuming, as well as leading to poor documentation.

The only guidance that 2167A provides with respect to the selection of CSUs is the definition of a CSU as being "separately testable"; additional clues might also be inferred from the information required in the SDD. This has caused some confusion among developers and customers because of different interpretations. Many have interpreted this to mean (at least initially) that a CSU is a subprogram or its equivalent, giving rise to an enormous amount of often unnecessary documentation. This interpretation is in conflict however with the requirements for configuration management after coding and unit testing [2167A 5.5.5]:

> *The contractor shall place the source code for each successfully tested and evaluated CSU under configuration control.*

implying that a CSU encompasses one or more source code entities which, at least in modern software development, will often comprise more than one subprogram. There is also a valid argument that a single program in a source file containing other subprograms cannot be genuinely "separately testable".

This is certainly an area of confusion but it is not one which will be satisfactorily resolved by searching standards for references as above. The important issue is that the design is documented to a satisfactory level and that both developer and customer (and the maintainer in some cases) agree as to the meaning of "satisfactory".

The authors believe that a CSU should map to one or more source files containing one or more subprograms or similar entities which are closely related. Examples that meet this criterion are:

- An Ada package (normally two or more source files).
- A small library of loosely related functions (such as a mathematical package).
- A group of data or type definitions.
- A database of limited complexity.
- One or more object class definitions.

Using this approach it is still likely that disagreements will occur as to the level of complexity allowed within a CSU. The developer's written software development method should address this issue, with additional written agreements with the customer as necessary.

Meyer et al. [MEY89], Ville et al. [VIL90] and McGann [MCG90] provide case studies of CSC and CSU selection for Ada projects.

7 DEVELOPMENT ISSUES

During the authors' survey of 2167A usage, both customers and developers highlighted a number of issues relating to the relationship between software development practices and 2167A. The issues can be categorised into two main areas: those associated with the overall developmental process and those associated with methods used during the process such as specific analysis and design methods.

7.1 The Development Process

A development process is the overall framework in which software is developed. This framework encompasses the various activities required during software development, such as analysis, design and testing. Several models have been proposed to help define this framework, the waterfall model being perhaps the most widely discussed (and criticised) of these.

Terminology tends to be a major stumbling block when discussing issues relating to software process models. Most discussions focus on the high-level or architectural models where people talk in terms of waterfall, spiral [BOE88], incremental, evolutionary and even whirlpool models [DEG90]. The situation becomes even more confusing if the many variations of the basic models are considered. For example, there are several variations of the waterfall model ranging from a strict interpretation (where the next phase cannot start until the previous phase is complete), to approaches where software is developed in a number of build increments (often termed incremental development). Statements such as "2167A is incompatible with evolutionary development" become meaningless because individuals, organisations and authors have different preconceived ideas as to what the term means. An incremental approach might rightly be termed evolutionary, yet others see evolutionary development as an approach where a prototype is evolved into a final product. Confusion abounds when this terminology is also used to describe acquisition processes. For example, how does evolutionary acquisition differ from evolutionary development? Clearly, the software world has once again fallen prey to overuse and misuse of terminology. Prior to any discussion of issues relating to software processes and 2167A, the terminology must be clearly defined. Discussions should proceed based on a consistent understanding of specific models, rather than ill-defined software jargon.

Applying a simple label such as "we are using incremental development" is insufficient to describe the overall process of software development. Published models may provide some general idea as to what is intended; however, they rarely fit well with a "real world" development. Programmers often remark that "the waterfall approach has been specified for our project, but it's not really the way we go about developing software". If this is the case, then what is the actual process or approach being used? How does it differ from the standard waterfall model? Has it been documented? How is the process being managed? Indeed, each project has specific characteristics and it would be inconceivable that a standard model could define the process for all projects. For example, an important characteristic that needs to be considered is risk. If a project requires the development of a complex user interface, then defining the requirements for the interface could be considered to be a high risk activity. In this case, user interface prototyping might be considered as an inclusion into the development process to assist in controlling the risk. Software development is a very complex endeavour. It is characterised by many features including tool usage, type of project, available resources, and possible risks. Reference to a simple high-level model is insufficient to describe the many complex interactions and dependencies that ultimately define the development process.

The 2167A standard is often criticised as being based on a waterfall approach. Although the waterfall approach is used as a model to help convey the requirements of the standard, it would

be foolish to believe that software development could proceed based solely on the model described in 2167A. In fact, the standard specifically states in para 4.1.1 that

> *The contractor shall implement a process for managing the development of the deliverable software. The contractor's software development process shall be compatible with the contract schedule for formal reviews and audits.*

It then goes on to list the major activities which can be applied "iteratively or recursively." Notice also that the standard states that the process is to be compatible with "the contract schedule". It does not specify when reviews, audits or deliverables are to be (or even have to be) provided. Indeed, these milestones need to be defined through close cooperation between the customer and developer. They will invariably be heavily dependent upon the process defined by the developer and agreed tailorings.

It is possible that the use of MIL-STD-1521B is more responsible than 2167A for forcing a project into a waterfall approach. There are also indications that some less disciplined software developers regard almost any controlled process as a waterfall approach because of the necessity for restrictions imposed by the control mechanisms (eg the need to undertake design activities before coding, and the need to document the design).

A major failing of many projects is that the development process is at best poorly defined. It is too easy to blame the "waterfall approach imposed by the standard" as the reason for failure; in most cases it is really due to a lack of planning and a lack of cooperation between the customer and developer. Without a clear direction as to how development will proceed, the project will become plagued with customer/developer disputes and unexpected surprises generally arising at the most crucial stages of development. Customers should ensure that the process to be used is well defined, understood, and documented. Interaction and dependencies between the various development activities should be provided and the model (as specified in the SDP) should be thoroughly reviewed and analysed.

The development process must be built around the fundamental requirements of 2167A, with particular regard to those requirements providing visibility to the customer and progress reporting. To propose a model without a thorough understanding of the requirements and implications of 2167A will almost certainly result in conflicts. There have been many arguments indicating that 2167A does not fit with the way the developer likes to develop software (generally based on some high level model). The authors contend that a process model needs to be elaborated for each project so that specific project characteristics can be accommodated. One of these is the tailored set of 2167A requirements. Humphrey [HUM89] proposes an approach based on process cells which would support elaboration of comprehensive project related process models. Indeed, his discussion of various aspects of process modelling is recommended reading. Regardless of the approach taken, those involved in developing a process model for a 2167A project must have a good understanding of the various approaches to software development, a solid background in process modelling, a knowledge of the project and product characteristics, and an expert knowledge of 2167A.

7.2 Development Methods

A requirement of 2167A (para 4.2.1) states that "the contractor shall use systematic and well documented software development methods to perform requirements analysis, design, coding, integration and testing of the deliverable software". No specific methods are defined, yet some methods have been found to be more easily documented using the DID format than others. Questions that arise include:

- Which methods should be used for a certain class of project?
- What constitutes a "well documented method"?
- What level of tailoring is required of 2167A DIDs for specific methods?

There are numerous published development methods - they range from academic approaches which lack rigour or scalability, to more rigorous and well documented methods developed by large organisations. In addition, there are several variations on a basic theme. For example, the words "object oriented" are used to describe a vast range of different methods. A characterisation of all the available methods (and their tailoring implications) is well beyond the scope of this report; the following guidelines may be of some assistance, however.

A common question is "which methods would most suit our project"? This begs the question "what are you building?" Prior to assessing the applicability of methods, the project must be characterised: is it real-time, hard real-time, data oriented?; how large and complex is it?; what are the major perceived risks? Other aspects to consider include process characteristics (eg "does the organisation intend using automated tool support?"), interrelationships with other methods (eg "will Structured Analysis fit well with an object oriented design?"). In addition to understanding project and process characteristics, the software engineers responsible for defining and selecting the methods must have a sound knowledge of available techniques and their application. In some cases, none of the available methods will ideally suit the project and elements of more than one method may need to be applied.

Regardless of the approach taken, the method must be well documented. Documentation should include:

- A description of the general philosophy and approach.
- A description of the graphical notations used.
- The basic steps to be taken.
- The method of recursion.
- Essential DID tailoring.
- Examples showing the products resulting from the application of the method.

Development methods must be defined early in the process. The methods to be used need to be thoroughly reviewed and assessed prior to software development. Some aspects that need to be considered include compatibility with other methods, compatibility with the overall development process, understandability, completeness, consistency and the effect on DID tailoring.

Analysis and design techniques cannot be left to the discretion of individual programmers; analysis and design activities should be conducted according to well defined and documented methods. These methods should be specified in the SDP at an early stage of the project, as required by 2167A, 2168 and AS-3563.

Clear separation of the analysis and design activities is critical (but is often difficult to achieve with inexperienced staff). In particular, the Software Requirements Specification (SRS) should contain requirements only, and avoid any software design information [SPR90, MCG90]. (This error occurs far too frequently and causes a protracted SSR, unduly volatile SRSs and increased documentation costs.)

Tailoring of the DIDs may be required to support specific methods. The degree of tailoring will depend on the method used and the project. For example, the SRS is arranged in terms of capabilities and separate data element requirements. Capability descriptions are functionally based and required details of inputs and outputs. If a Structured Analysis approach is used, little tailoring of the SRS would be required since the DID contents and format correspond to

this type of approach. The use of an object oriented analysis technique, however, may require significant tailoring. Here the aim is to document the analysis in terms of objects, where an object may encapsulate both data and operations. This approach has been used very successfully for a range of projects and should not be dismissed simply because it does not fit well with the DID structure. If significant DID tailoring is required, the customer and developer must work together for the best solution. The developer must have a well defined method, educate the customer in the method and proposed tailoring, and show how the approach best suits the particular development. On the other hand, the customer must be receptive to new thinking and realise that there is a likelihood that the DIDs will require tailoring for specific methods. The most important thing is that the customer and contractor must collaborate in reaching an agreed tailoring.

8 DOCUMENTATION ISSUES

8.1 General recommendations

The documentation required by the 2167A DIDs is designed to be an integral part of 2167A developments. Not only does each document complement the others, but each has its place in the development process. Ignorance of the relationships between the documents and each document's role in the process can lead to serious increases in documentation costs as well as other problems in the development process. Lack of understanding and agreement between the customer and developer as to what is required in the documentation will also lead to unnecessary work and delays (see also Section 5.2).

One simple example of the need to consider each document's contents in the framework of the documentation family is the definition of the software test environment in the Software Test Plan (STP). In most projects the test environment is a subset of the software engineering environment described in the SDP and is best explained in that context. It therefore makes sense in these cases to describe the test environment in the SDP at the level of detail required by the STP and to refer to it in that document to prevent duplication.

DOD-STD-2167A requires that a systematic development method be used and the DIDs reflect this. More importantly, the DIDs assume, rightly or wrongly, a structured analysis and design process based on functional decomposition. If the actual method used does not follow this broad model the mapping of the analysis and design to the DIDs will be at best difficult and at worst impossible, causing problems both in the preparation of documents and in their acceptance by the customer. In particular, if the analysis step is not performed (usually an indication of the developer not following a systematic method) the SRS and IRS (Software and Interface Requirements Specifications) will be difficult to write and will be almost valueless.

General guidelines for the preparation of 2167A documentation are as follows:

a. Always consider each document's role in the documentation family and the development process.

b. Consider the potential audience of each document.

c. Consider the costs involved in changing baselined documents when writing them.

d. Provide no more information in each document than is necessary. If additional information could be useful, include it as "information only" in an appendix or as a

separate document. Avoid the temptation of dumping all the information currently available into a requirements or design document.

e. Avoid duplication which may lead to inconsistencies when changes are made. In many cases this can be achieved by cross-referencing.

f. Avoid the documentation of design details in requirements documents.

g. Ensure that the development method and CASE tools used are compatible with the requirements of 2167A and the development of 2167A documentation.

8.2 Documentation on electronic media

The DIDs are not prescriptive about the form in which the documents are to be delivered. Each DID states:

> *7.2 The Contract Data Requirements List should specify whether this document is to be prepared and delivered on bound 8 1/2 by 11 inch bond paper or electronic media. If electronic media is selected, the precise format must be specified.*

While the authors are not aware of projects in Australia where the documentation is being delivered only in the form of electronic media, it is sometimes provided in addition to the printed form.

There are advantages in using electronic media for documentation including:

- Changes may be quickly made to the documentation.
- The documentation may be transmitted by electronic means. This is an advantage not only in the initial delivery but also for distribution among geographically separate members of the customer's team.
- Different versions of the same documents may be compared easily to highlight changes.
- Document users may use software tools to scan documents for key words.
- Storage of documentation, particularly multiple versions, is more efficient.
- Costs and response time in the generation and distribution of documentation, particularly where a large number of copies might be normally required, may be reduced.
- The customer may more effectively maintain the documentation throughout the life of the system.

There are also disadvantages:

- The configuration control of documentation needs special attention if its most common form is electronic.
- Electronic documentation may result in additional direct costs to the customer, both in the procurement of compatible documentation equipment and in the training of

staff (although this may be reduced or eliminated by the specification of a documentation system already in use in the customer organisation).

The authors recommend that customers require the delivery of documents in electronic form even if printed copies are also required. Requirements for printed copies should take into consideration the availability of electronic copies and be reduced accordingly.

In determining the acceptability or otherwise of the format of documentation on electronic media, the following should be taken into consideration:

- What facilities will be required to use the documentation and to make printed copies. Consideration should include other customer agencies which may need to use the documentation.
- Whether the format will allow the customer to make changes to the documentation such as inserting comments, or updating documents during software maintenance.
- Whether the format and tools support comparison of document versions.
- Whether the format supports documentation in graphic form.
- The physical medium, if any, for delivery.

For Defence projects, the customer should also ensure that documentation on electronic media conforms with the CALS requirements (Computer-aided Acquisition and Logistics Support).

8.3 Tailoring the DIDs

In tailoring the requirements for documentation, the aim should be to produce the minimum consistent set of documents which meets the needs of the project.

8.3.1 General

The deletion of individual DIDs from the documentation requirements for a project is often desirable but consideration needs to be given to the full consequences of its deletion.

If the customer intends to rely on the developer to provide support for the software, for example, there may be no need for a Computer Resources Integrated Support Document (CRISD), Computer System Operator's Manual (CSOM) or Software Programmer's Manual (SPM) to be produced. Customers should be aware, however, that there are several levels of support which may be required. Modern software is often configurable and the reconfiguration may be quite complex. By deleting the requirements for the CRISD and SPM the normal vehicle for the supply of this information is removed (see Section 9.8).

Deletion of a high level DID, such as the SSDD, will also require the tailoring of several other DIDs. The table below shows the relationship between the various DIDs and standards, indicating references from one document to another. Although not all of the references are significant (in some cases they are mainly for information purposes) the table shows that care must be taken in deleting DIDs.

One misapprehension that the authors have noticed in several projects is the view that 2167A is all-encompassing and that no additional documentation requirements are necessary. While this may be true in some cases, many projects require additional documentation which should be

identified in the CDRL, either as the tailoring of specific DIDs or as separate documents (see also section 2.6). Furthermore, good software engineering practice always requires the generation of diagrams or other information not explicitly specified in any DID. Examples include user interface layouts, concurrency diagrams and timing analyses. These should all be provided somewhere in the delivered documentation.

8.3.2 Adaptive tailoring

In many projects a single tailoring for all SRSs and SDDs (say) will not be adequate. Different requirements for the software development will result in the need for different tailorings in some cases. It will also be necessary in some circumstances to apply different tailorings for different CSCs or CSUs, for example when there is a combination of new and existing software in a CSCI.

8.3.3 Using alternate formats in DIDs

The DIDs are deliberately prescriptive with regard to the structure (including paragraph numbering) and content of documentation. This approach is intended to guarantee the

	Documents referenced																					
	SSS	PIDS	CIDS	490	483	SDP	SSDD	IRS	SRS	IDD	SDD	STP	STD	STR	SPS	VDD	CSOM	SUM	SPM	FSM	CRISD
SDP	+	+	+																		
SSDD	+																				
IRS				+	+					+											
SRS	+	+	+	+	+			+													
IDD				+	+			+			+										
SDD				+	+			+	+	+					+						
STP						+		+	+												
STD								+	+			+									
STR													+								
SPS				+	+						+					+					
VDD					+																
CSOM																					
SUM																	+				
SPM																	+				
FSM																					
CRISD																					

Cross-referencing between DIDs and standards

completeness of the documents and to assist reviewers in finding specific information. Any consideration of tailoring or alternative formats must take this into account.

While the DIDs control the content and layout of documentation they encourage the presentation of information in formats appropriate to the content, and the use of cross-referencing rather than duplication. Statements such as the following [SDD 10.1.6.2.2] are relatively common:

> *This information may be provided by automated tools or other techniques, such as a program design language, flowcharts, or other design representations.*

The authors also recommend the use of appendices or separate documents where this will improve the presentation and use of information. This can be done in many cases without the need for tailoring as long as the basic structure of the document remains intact and coherent, the extracted information is at a relatively low level (eg paragraph 4.X.Y.2 in the SDD which addresses the design of an individual CSU), and the subsequent references are direct and unambiguous. Such a practice is particularly useful for bulky material, supporting information and the documentation of information which is orthogonal to the basic structure of the document.

The more general extension of this practice, ie the use of DIDs as shell documents with (often broad) references to documents written to the developer's internal standards, should be avoided.

8.4 Software development files

Developers are required to "document and implement procedures for establishing and maintaining SDFs" [2167A 4.2.9]. The authors view SDFs (software development files) as very important for the documentation of additional development information including justification of design choices and alternative considerations, CSU test procedures, and other information, sometimes of an informal nature, that may assist in software maintenance. In addition, if the SDD is tailored to the extent suggested in Section 9.5, the SDFs are critical for the documentation of lower level CSUs and the support of SDD documentation during CDRs. This approach follows that of Springman [SPR89].

The SDFs should be placed under configuration control following the successful testing of their associated software element (normally CSU or CSC) and be delivered to the customer if the transitioning of software support is required.

DOD-STD-2167A should be tailored to include these requirements. The customer should also ensure that the developer has adequate procedures for the use and maintenance of SDFs, and that the CDRL includes their delivery. It may be necessary in some circumstances to negotiate the protection of intellectual property contained in delivered SDFs.

9 THE DATA ITEM DESCRIPTIONS (DIDS)

9.1 Software Development Plan

The SDP describes the organisation, management and planning of the software development for a project.

The effort required in preparation of this very important document can normally be reduced by referencing the developer's own software development standards wherever possible. Where this

occurs those standards must be provided to the customer for the review of the SDP, and must be available to the development team before development starts. It may be preferable (for both customer and developer) to describe the high level requirements being met in the SDP and to use references to show how those requirements are met.

One area of definition in the SDP which will vary greatly between different projects is *Risk management* (SDP section 3.3). Consideration should always be given to tailoring this section for the specific application and project.

9.2 System/Segment Design Document

The SSDD describes the system (or segment), the allocation of CSCIs and HWCIs and the processing resources. Its definition assumes that its requirements are derived from a System/Segment Specification (SSS) as defined under MIL-STD-490A and it should be tailored if the system specification is a Prime Item Development Specification (PIDS), Critical Item Development Specification (CIDS) or other specification.

Where there is only one CSCI and the operational requirements are well defined, the SSDD may not be necessary in its entirety. In this case it may be tailored out, and any of the SSDD requirements that are considered relevant may be added to the SRS.

When preparing system specifications, it is important that customers aim to avoid restricting the design of the system. Over-specification of requirements can make the analysis and design task more difficult, as well as forcing design details into requirements specifications such as the SRS.

9.3 Interface documents (IRS and IDD)

The authors' survey revealed a level of dissatisfaction with duplication between the IRS and IDD and the usefulness of both documents. In the authors' experience these documents are often misunderstood and subsequently misused. The wording of the DIDs does not assist in this regard.

An IRS is used to specify the requirements for one or more CSCI's external interfaces, the details of which are necessary for the preliminary and detailed design of the system. Interfaces may be to other CSCIs, HWCIs or external to the system. After SSR the IRS becomes part of the allocated baseline. The level of information provided in the IRS should reflect the interface's criticality in the design of CSCIs. Detailed interface information should only be specified for interfaces which are external to the system, whose definition is critical to the design of CSCIs, or which are defined and are unlikely to change (such as the interface to an existing HWCI). Other interfaces should be identified and specified in terms of their requirements. It should contain no design information.

To some extent this is a trade-off. Information that is likely to change during design should not be in an IRS - as it is part of the allocated baseline changes can be costly. On the other hand the design of a CSCI requires the definition of external interfaces, and instability in its external interface definitions is a source of risk. The customer and developer must therefore agree on which interfaces must be defined in the IRS and which must be "designed" and expanded in the IDD.

The IRS and IDD can therefore be regarded as containing the appropriate level of interface definitions at PDR and CDR respectively. The IDD, being a derivative of the IRS, must inevitably duplicate some of the information contained therein, but this problem will be reduced

if the IRS contains the correct level of information and there are no major instabilities in the overall system design. It can be further reduced by referencing the IRS from the IDD, where the IRS contains detailed information, although it may be preferable to include all the interface information in one document, which should be the IDD.

In many smaller projects a single interface document will suffice. This will either be the IRS or IDD depending on the relevant stability of interfaces external to the system and those between CSCIs. In these cases tailoring must be used to delete the requirements for the discarded document, and also to compensate for its absence. The configuration management requirements must also be tailored accordingly.

There are also situations where neither an IRS nor IDD is needed. Where a project consists of a single CSCI running as a program in a single computer, where there are no specific interfaces between the CSCI and the computer (the only HWCI), and where there are no external interfaces or interfaces to other configuration items, there is almost certainly no need for an IRS or IDD. There are also projects where some or all of these requirements are not met and the required interface information can be provided satisfactorily in the SSDD or an SRS, provided that these documents are tailored to provide an adequate level of detail.

9.4 Software Requirements Specification

The SRS is usually the key contractual specification for software development, particularly when the software is being developed by a subcontractor.

In the authors' experience the two most common problems in SRS preparation are the absence of a genuine analysis of the requirements and the inclusion of design information which should instead be in the SDD. Often these problems occur together when inexperienced development staff confuse the analysis task with that of preliminary design.

An SRS should only contain requirements information which stems from a higher level statement of requirement (SSDD, SSS or other specification) and which is refined as a result of the developer's requirements analysis. Information that is likely to change as a result of detailed design should either be relegated to the SDD or included separately on an "information only" basis.

The authors have noticed a reluctance by some developers to derive additional requirements from those specified, resulting in an SRS which is little more than a reflection of the higher level specifications. This practice is to the disadvantage of both the customer and the developer [BUL88] and should lead to rejection of the SRS. The customer should be seeking an indication from the SRS of the developer's understanding of the requirements. Approval of the SRS means that the derived requirements are then included in the allocated baseline and the developer can proceed to design with an approved set of consistent and detailed requirements.

Another potential problem is the attempt to map the results of an inappropriate analysis method onto the structure of an untailored SRS. As discussed in Section 3.3, it is important that a developer's analysis and design methods are consistent with the development framework described by 2167A. If the requirements of the SRS and SDD documents are ignored it is likely that the documentation task will be difficult and the result will be of little value. Developers should be wary of the advice given by Coad and Yourdon [COA90]: "You can follow good principles of analysis and design, and then figure out how to fit your results into the 2167A framework." They would be better advised to plan how their methods will fit the formats of the DIDs and to tailor their methods and the DIDs accordingly.

The main tailoring for an SRS is likely to be to accommodate different analysis methods. This is discussed in Section 7.

9.5 Software Design Document

The SDD is used to document the design of a CSCI and its partitioning into CSCs (including sub-level CSCs if required) and CSUs, which relate directly to the actual source code. It is unfortunate that the SDD is often viewed (at least in the short term) only as a milestone to be passed at CDR. Any tailoring of this document should consider its use during the development and maintenance of the system.

It can be inferred from the requirements of 2167A and the SDD that a CSU is a single and separate source code element (see also Section 6.2), containing a single entry point and no data definitions whose scope extends outside the unit; and that all data accessed from more than one CSU is global to the CSCI. No information is required on the scope or visibility of code or data elements.

The authors believe that this interpretation of a CSU and the SDD's treatment of data is incompatible with many existing design methods and modern programming languages. For example, it will not accommodate the adequate documentation of the following:

- The definition of related data and subprograms in an Ada package.
- The definition of object classes or data types as separate program elements.
- Elements such as Ada tasks and generic packages.

In addition, the level of description required [SDD 4.X.Y.2], if applied to all subprograms, is more than is normally needed for development, maintenance or visibility of the design by the customer.

In the authors' opinion, the SDD in its untailored form is unsuitable for almost any project. It is therefore strongly recommended that the SDD be tailored for all projects depending on the application, the design method and the language used (among other factors).

The most difficult issue facing the customer and developer in the tailoring of the SDD is not seen as the definition of the structure of the document (or documents), or requirements for the documentation of specific software elements, but in the coherence of the design description and the level of detail supplied. While the level of detail required in the *Design* paragraph may be excessive for many units, it will be necessary to define some or all of this information in certain circumstances. Determining firm generic criteria acceptable to both customer and developer for the different design detail required for different CSUs is almost impossible. (It is probably this fact that is responsible for the current rigid and comprehensive - some would say draconian - requirements of the SDD.)

It is therefore suggested that the tailoring should include the form and content of the documentation required for different types and levels of CSUs, where comprehensive guidelines are provided for the selection of the level and hence the detail required in their definition. It is important that this tailoring be agreed prior to contract signing. Approval or otherwise of the choice of level would ultimately rest with the customer, as part of the design approval. The risk that the customer will not approve the levels chosen should be reduced by informal discussions between the customer and developer prior to the commencement of the

documentation of detailed design. These may include written agreements with regard to the content of the SDD.

The above approach will only work successfully if the customer has the capability to understand the design and the risks and advantages in the different levels of documentation. Such a capability requires appropriate skills and experience, coupled with regular discussions with the developer throughout the design process (Section 3.2).

9.6 Software Product Specification and Version Description Document

The Software Product Specification (SPS) in its untailored form provides four items of information: a reference to the relevant SDD, the source listings or a reference to them, compiler/assembler identification and the measured resource utilisation. It forms part of the product baseline.

The authors see little value in the provision of source listings for modern software projects if the source software is provided in electronic form. Without source listings, the SPS is relatively insubstantial and, unless there are reasons for providing additional information in the SPS (by tailoring), it may be tailored out. The remaining requirements, if appropriate, may be added to the Version Description Document (VDD). It is important, however, that the updated version of the SDD be included as part of the product deliverables (in lieu of its inclusion in the SPS), and additional tailoring will be necessary to ensure this.

9.7 Test documents

If formal qualification testing is required, the Software Test Plan (STP), Descriptions (STDs) and Reports (STRs) will be required.

While the content and format of these documents is adequate in many cases, the test descriptions can often be described more effectively and efficiently using a format which is specifically developed for the application being tested or the test method (eg in automated testing). In these cases, however, the principles of the untailored DID should be followed in formulating test descriptions that are unambiguous, repeatable and traceable to the requirements that they test. Care should also be taken to avoid excessive duplication which may lead to errors when changes are made.

Test results will come in various forms dependent on the application and test tools. It is important that these be accepted in their output form (eg printouts, test logs), rather than transcribing them into an unsuitable format in an STR. The STR requirements will need to be tailored to accommodate these changes.

9.8 Computer Resources Integrated Support Document

The CRISD requirements will generally be found to be too broad to provide adequate documentation of the transitioning requirements. This DID should be tailored to suit specific projects and their objectives for software support.

Customers may find DOD-STD-1467(AR), Software Support Environment, useful in tailoring the CRISD. Although poor in terms of readability, this standard provides additional requirements covering:

- the use of and compatibility with existing support facilities,
- software rights

- catering for different categories of software (such as COTS)
- the minimum requirements for support facilities, and
- advice with regard to qualification testing of the support facilities.

9.9 Manuals (CSOM, SUM, SPM and FSM)

The Computer System Operator's Manual (CSOM), Software User's Manual (SUM), Software Programmer's Manual (SPM) and Firmware Support Manual (FSM) provide general requirements which, in the authors' opinion, are unlikely to lead to satisfactory useable manuals. The format and content of manuals will usually be strongly dependent on the application and its environment. Each of these DIDs, if required, should be specifically tailored for its intended use.

Defence customers will often require their own standards for manuals, which will override the format requirements for these DIDs. This should not necessarily result in the DIDs being discarded, however. They may still serve a purpose by specifying the structure, content and level of description of the required information.

It should be noted that the Software Programmer's Manual, despite its attractive title, is rarely necessary in modern software development projects. It is, in essence, an assembly language manual for the computer/s used.

10 FURTHER WORK

The authors recognise the fact that this study has examined only some of the aspects relevant to software development using DOD-STD-2167A, and has concentrated on what, in their opinion, are the most pressing problems. Exposure to these problems has, however, identified several other areas where further effort is likely to assist in the management of 2167A projects, the communication between customers and developers and the maintainability of software so produced.

10.1 Assessment of developers and customers

It has become evident to the authors that some developers in Australia may not have the knowledge, experience or standards to develop software in accordance with 2167A. Similar deficiencies in customer's project teams indicate that their ability to manage software development projects is in question. Guidelines need to be prepared to enable customers to assess the capability of potential developers and their own project staff to participate in a 2167A project.

10.2 Preparation of RFTs and tenders

The authors' survey and workshop revealed that developers often have difficulty in the preparation of tenders, in their understanding of what is really wanted and how the tenders will be interpreted by the customer. Some are reluctant to suggest significant tailorings of 2167A, for example, even when the RFT encourages tailoring, for fear that their tender will be rejected on that basis. Research is required into the contents of RFTs and tenders, to recommend procedures that will improve the cost and quality of the software product.

10.3 Electronic documentation and communication

It is inevitable that there will be an increasing reliance on electronic documentation and communication in Defence projects. Without standardisation this will result in major costs and disarray, with the possibility of reduced communication between developers and customers in the short term. The Defence policy to standardise on CALS should assist in this regard, but research is needed to investigate the best use of CALS in software developments and to make appropriate recommendations

10.4 Tailoring for internal Defence projects

Software developed within Defence includes maintenance of existing systems, development of new systems and prototypes, and a large range of different types of software development within DSTO. In many cases software is required to be developed in accordance with 2167A.

Internal development differs from development by contract mainly in the level of trust placed in the developer. In some cases, particularly in system support centres for existing software, the developer may also be responsible for the generation of system level specifications and for overseeing formal qualification testing. In others, such as when DSTO is responsible for the development of operational software, the customer may have a more pronounced role, but is unlikely to require the level of visibility regarded as necessary for an external development.

Because of the different requirements for visibility of the design and review of progress, the tailoring for such projects will be different from those for external developments. Guidelines need to be developed for the tailoring of 2167A in these circumstances.

10.5 DOD-STD-2167A resources

The authors see a need for a collection of resources in Australia assisting customers and developers in the education and use of 2167A. These should include:

- A comprehensive library and bibliography of relevant information relating to 2167A projects
- Definitions and interpretations of 2167A and DID requirements.
- Sample tailorings for different project types.
- Word processor templates for 2167A documentation.

10.6 Guidance for different applications and development methods

Further work is required to determine detailed tailoring guidelines for different application types and development methods. Applications should include real-time embedded systems, distributed systems, information systems and safety or security critical systems. Development methods should include incremental and phased developments, object oriented analysis and design techniques, and the use of mathematically based ("formal") methods.

10.7 Guidance for the use of V&V in 2167A projects

Our survey indicated uncertainty in the use of Verification and Validation in Defence projects, both in the application of V&V in the development process (internal or independent) and in the monitoring of the V&V activities by the customer. Further work is required to investigate

V&V standards, the value of independent V&V, and suitable frameworks for interfaces between the various V&V staff and the development team.

10.8 Follow-up studies

Although DOD-STD-2167A is a development standard, it will have effects on the support of software for many years after development. As yet these effects cannot be confidently assessed. The results and recommendations of this study should be reviewed in two to three years time in the light of further experiences, particularly with regard to the support of software developed to the requirements of 2167A.

10.9 Update for DOD-STD-2167B

The US DoD software standards community is developing the next version of DOD-STD-2167 under the working title of MIL-STD-SDD (for Software Development and Documentation). It is expected that this standard will combine (harmonise) the requirements of DOD-STD-2167A and DOD-STD-7935. Apart from any contributions which may be made to this endeavour, such as the results of this study, there is likely to be a need to analyse the new standard for use in Australian software developments.

11 CONCLUSIONS

The course of DOD-STD-2167A software development projects in Australia has not generally been smooth, but neither was the course of such projects prior to the introduction of the standard. There are, however, specific problems which have been highlighted by the use of 2167A.

The prime problem, which is a major contributor to most of the other difficulties, is the lack of experience and understanding of customers and developers both in the use of 2167A and in software development management. This deficiency has led to many other problems including inadequate and inappropriate tailoring of the standard, poor or even unusable documentation, cost and schedule overruns, and software products of low quality. Another serious problem is the lack of communication between customers and developers in development projects.

It is hoped that the recommendations and advice in this paper will help to improve this situation, in making customers and developers aware of the difficulties they face, and providing some guidance in their avoidance or resolution.

The most important recommendation, however, is that both customers and developers must dedicate more effort to the education of their staff, so that they are more able to cope with the development of complex software systems to a standard which does not forgive ignorance or amateurism.

REFERENCES

U.S. MILITARY STANDARDS

[248A] DOD-HDBK-248A, "Guide for Application and Tailoring of Requirements for Defense Material Acquisitions", October 1979.

[287] MIL-HDBK-287, "A Tailoring Guide for DOD-STD-2167A, Defense System Software Development", August 1989.

[480B] MIL-STD-480, "Configuration Control - Engineering Changes, Deviations, and Waivers", July 1988.

[481B] MIL-STD-481, "Configuration Control - Engineering Changes, Deviations, and Waivers (Short Form)", July 1988.

[490A] MIL-STD-490A, "Specification Practices", June 1985.

[483A] MIL-STD-483A, "Configuration Management (CM) Practices for Systems, Munitions, and Computer Software", June 1985.

[499A] MIL-STD-499A, "Engineering Management", May 1974.

[1521B] MIL-STD-1521B, "Technical Reviews and Audits for Systems, Equipments, and Computer Software", June 1985.

[1467A] DOD-STD-1467(AR), "Software Support Environment", January 1985.

[2167A] DOD-STD-2167A, "Defense System Software Development", February 1988.

[7935A] DOD-STD-7935A, "DOD Automated Information Systems (AIS) Documentations Standards", October 1988.

DOD-STD-2167A DATA ITEM DESCRIPTIONS (DIDS)

DI-MCCR-80030A	Software Development Plan
DI-CMAN-80534	System/Segment Design Document
DI-MCCR-80026A	Interface Requirements Specification
DI-MCCR-80025A	Software Requirements Specification
DI-MCCR-80027A	Interface Design Document
DI-MCCR-80012A	Software Design Document
DI-MCCR-80014A	Software Test Plan
DI-MCCR-80015A	Software Test Description
DI-MCCR-80017A	Software Test Report
DI-MCCR-80029A	Software Product Specification
DI-MCCR-80013A	Version Description Document
DI-MCCR-80018A	Computer System Operator's Manual
DI-MCCR-80019A	Software User's Manual
DI-MCCR-80021A	Software Programmer's Manual
DI-MCCR-80022A	Firmware Support Manual
DI-MCCR-80024A	Computer Resources Integrated Support Document

AUSTRALIAN STANDARDS

[3563] AS 3563-1988, "Software Quality Management System", August 1988.

[3901] AS 3901-1987 (ISO 9001-1987), "Quality Systems for Design/Development, Production, Installation and Servicing", December 1987.

OTHER REFERENCES

[BOE88] Boehm B.W., "A Spiral Model of Software Development and Enhancement", IEEE Computer, pp. 61-72, 1988.

[BUL88] Buley E.R., Moore, L.J. and Owens, M.F., "B5 (SRS/IRS) Specification Guidelines", ESD-TR-88-337, USAF Electronics Systems Division, Hanscom Air Force Base, MA, December 1988.

[COA90] Coad P. and Yourdon E., "Object-oriented Analysis", Prentice Hall, 1990.

[DEG90] DeGrace P. and Stahl L.H., "Wicked Problems, Righteous Solutions: A Catalogue of Modern Software Engineering Paradigms", Prentice-Hall, 1990.

[FIS87] Fisher R. and Ury W., "Getting to YES - Negotiating Agreement Without Giving In", Arrow Books, 1987.

[GAB91] Gabb A.P., Pollard P.C., Landherr S.F., Vernik R.J., "Tailoring DOD-STD-2167A - A Survey of Current Usage", WSRL-TN-57/91, December 1991.

[HUM89] Humphrey W., "Managing the Software Process", Addison-Wesley, New York, 1989.

[LOG90] Logicon, Inc., TAILOR/2167A Version 2.2, TAILOR/DIDs-2167A Version 1.0, INSIGHT/2167A Version 1.0, CDRL-GEN Version 2.0, 1990.

[MAR88] "Using Ada with DOD-STD-2167A". Martin Marietta Information and Communication Systems, December 1988.

[MCG90] McGann R.J., "The Application of US DOD-STD-2167 to Real Time Ada Projects: Some Lessons Learnt", Proceedings, The Fifth Australian Software Engineering Conference, May 1990.

[MEY89] Meyer C.A., Lindholm S.C. and Jenson J.L., "Experiences in Preparing a DoD-STD-2167A Software Design Document for an Ada Project", Proceedings, TRI-Ada '89, October 1989.

[OVE90] Overmyer S.P., "The Impact of DoD-STD-2167A on Iterative Design Methodologies: Help or Hinder?", ACM SIGSOFT, Software Engineering Notes, vol 15 no 5, October 1190.

[SPR89] Springman M.C., "Software Design Documentation Approach for a DOD-STD-2167A Ada Project". Proceedings, TRI-Ada '89, October 1989.

[SPR90] Springman M.C., "Incremental Software Test Approach for DOD-STD-2167A Ada Projects", Proceedings, TRI-Ada '89, October 1989.

[VIL90] Ville C. and Bratel A., "A Real Time Ada Design Method Based on DOD-STD-2167A", Proceedings, TRI-Ada '90, December 1990.

DISTRIBUTION

	Copy No.
Defence Science and Technology Organisation	
Chief Defence Scientist)	
Central Office Executive)	1 shared copy
Counsellor, Defence Science, London	Copy of Doc Cont Data Sht
Counsellor, Defence Science, Washington	Copy of Doc Cont Data Sht
Scientific Adviser, Defence Central	1
Scientific Adviser, Defence Intelligence Organisation	1
Navy Scientific Adviser	1
Air Force Scientific Adviser	1
Scientific Adviser, Army	1
DSR - Bangkok	Doc Cont Data Sht
SA to DRC - Kuala Lumpur	Doc Cont Data Sht
Director, Aeronautical Research Laboratory	1
Chief, Aircraft Structures and Materials Division	1
Chief, Aircraft Systems Division	1
Chief, Guided Weapons Division	1
Director, Surveillance Research Laboratory	1
Chief, Microwave Radar Division	1
Chief, High Frequency Radar Division	1
Director, Materials Research Laboratory	1
Chief, Maritime Operations Division	1
Chief, Explosives Ordnance Division	1
Electronics Research Laboratory	
Director	1
Chief, Information Technology Division	1
Chief, Communications Division	1
Chief, Electronic Warfare Division	1
Head, C3I Systems Engineering Group	1
Head, Command Support Systems Group	1
Head, Trusted Computer Systems Group	1
Head, Information Acquisition and Processing Group	1
Mr A.P. Gabb, C3I Systems Engineering Group (Author)	5
Mr P.C. Pollard, C3I Systems Engineering Group (Author)	70
Head, Software Engineering Group (S.F. Landherr (Author))	6
Mr R.J. Vernik, Software Engineering Group (Author)	4
Publications & Publicity Officer, Information Technology Division	1
Media Services	1
Department of Defence	
Assistant Chief of the Defence Force (Development)	1
Assistant Chief of the Defence Force for Logistics	1

Assistant Chief of the General Staff - Logistics	1
Assistant Chief of the General Staff - Materiel	5
Assistant Chief of the Air Staff - Materiel	5
Assistant Chief of Naval Staff - Materiel	5
Deputy Secretary, Acquisition and Logistics	1
First Assistant Secretary, Capital Equipment Programs	1
Assistant Secretary, Communications Planning Branch, PDC	5
Director General, Communications and Information Systems	5
Director General, Logistics Policy	1
Air Force Office	
Director, Electronics Engineering, Air Force	5
Army Office	
Director, Command and Control Procurement, Army	1
Acting Superintendent, Software Systems Engineering Division, EDE	4
Superintendent, Communications Division, EDE	1
Navy Office	
Director, Navy Combat System Engineering	5
Libraries and Information Services	
Australian Government Publishing Service	1
Defence Central Library, Technical Reports Centre	1
Manager, Document Exchange Centre, (for retention)	1
National Technical Information Service, United States	2
Defence Research Information Centre, United Kingdom	2
Director Scientific Information Services, Canada	1
Ministry of Defence, New Zealand	1
National Library of Australia	1
Defence Science and Technology Organisation Salisbury, Research Library	2
Library Defence Signals Directorate, Melbourne	1
Australian Defence Force Academy Library	1
British Library Document Supply Centre	1
Spares	
Defence Science and Technology Organisation Salisbury, Main Library	10
Total number of copies	183 copies

Department of Defence

DOCUMENT CONTROL DATA SHEET

Page Classification	
UNCLASSIFIED	
Privacy Marking/Caveat (of document)	
UNCLASSIFIED	

1a. AR Number	1b. Establishment Number	2. Document Date	3. Task Number
AR-006-979	ERL-0637-RE	SEP 92	DST 89/218

4. Title	5. Security Classification	6. No. of Pages	38
RECOMMENDATIONS FOR THE USE AND TAILORING OF DOD-STD-2167A	U (Document) U (Title) U (Abstract)	7. No. of Refs.	29

S (Secret) C (Confi) R (Rest) U (Unclass)

* For UNCLASSIFIED docs with a secondary distribution LIMITATION, use (L) in document box.

8. Author(s)	9. Downgrading/Delimiting Instructions
A.P. Gabb, P.C. Pollard, S.F. Landherr, R.J. Vernik	N/A

10a. Corporate Author and Address	11. Officer/Position responsible for
Electronics Research Laboratory PO Box 1500 SALISBURY SA 5108	Security..... Downgrading..... Approval for Release.....DERL.....
10b. Task Sponsor DSTO	

12. Secondary Distribution of this Document

APPROVED FOR PUBLIC RELEASE

Any enquiries outside stated limitations should be referred through DSTIC, Defence Information Services, Department of Defence, Anzac Park West, Canberra, ACT 2600.

13a. Deliberate Announcement

No Limitation

13b. Casual Announcement (for citation in other documents)

[√] No Limitation

[ ] Ref. by Author , Doc No. and date only.

14. DEFTEST Descriptors	15. DISCAT Subject Codes
Standards, Software engineering	1205

16. Abstract

This report is the culmination of a study into the use of DOD-STD-2167A in Australian software development projects. It makes recommendations for the use and tailoring of the standard.

Doc Sect WF11

Page Classification
UNCLASSIFIED

16. Abstract (CONT.)

17. Imprint

Electronics Research Laboratory
PO Box 1500
SALISBURY SA 5108

18. Document Series and Number	19. Cost Code	20. Type of Report and Period Covered
ERL-0637-RE	329/796917	REPORT

21. Computer Programs Used

N/A

22. Establishment File Reference(s)

N/A

23. Additional information (if required)